Cプログラミング1(再)
第9回

講義では、「Cプログラミング入門以前」を教科書として学び
演習では、実際にCのプログラムを書く、実行する、レポートを書くことで学ぶ

9章　関数

Cプログラムは、

　　必ず１つある main関数と

　　その他の複数(ない場合もある)の関数

によって構成される

だから、関数を作れることがとても大事

9.1 関数とは？

9.1.1 数学の関数

例: $y = f(x)$ ただし $f(x)=x^2+2x-1$

「yは関数f(x)で表される」

ここに現れる引数 x：「媒介変数(パラメタ)」

xの値によってどのようにyが求まるかの記述

9.1.2 Cの関数

関数には2種類

値を返す関数 ---- 関数定義において返す値の「型」を宣言

値を返さない関数---- 関数定義において void 型を宣言

値を返す関数の例: ---- 関数の定義で return により返す値を決定

```
c = getchar()
y = sin(x)
```

値を返さない関数 --- return を使っても良いが引数を書かない

```
exit(0)
abort()
```

9.1.3 関数を作る理由

原則：

賢い人ほど、たくさんの関数を作っておき、新しい仕事をするにはそれらを組み合わせてできないかをまず考える

1) 関数の内部を知らなくても、利用しやすいようにする

2) 似たような処理を関数の形でまとめ、利用する---プログラムが短く、わかりやすくなる

3) 他のプログラムでも再利用できるようにする---プログラム作成の手間の省力化

コラム: 関数の作成と文書化

関数を使ったり、拡張するときには、その関数の取扱説明書（文書）がないと使いにくい（使えない）

だから、自分で新しい関数を作る場合には、つぎのことを書くようにしておこう（プログラムの先頭にコメントの形で書くのが良い）

1. 入力（どのような型か、それが表すデータの意味）
2. 出力（戻り値、どのような型か、何を表すか）
3. 処理内容(どのようなアルゴリズムか）
4. 注意点（潜在的なバグの存在、見落しの可能性、など）

9.2 関数の基礎知識

9.2.1 メインルーチンとサブルーチン

主役:メイン、サポート役(下請け): サブ

9.2.2 引数と戻り値

関数の呼び出し

関数の戻り

実引数

仮引数

nの値をコピーし局所変数xの値にセット

```
int main(int argc, char *argv[]) {
    int n,x;

    n = atoi(argv[1]);

    x = factorial(n);

   printf("%d! = %d¥n" n,x);
}
```

```
int factorial(int x) {
    int y=1;
    ...
   ...
    return y;
}
```

関数が呼ばれた場所に制御を戻す
returnに書かれた値で関数呼び出しの値とする

9.2.3 関数を使うときには宣言が必要

関数の定義の例:

```
int   factorial (int n)
float pow (float x, float y)
```

戻り値の型　　引数の型

関数の呼び出しの文の後で関数定義が来る場合
関数のプロトタイプ宣言が必要。その場合、上の情報が必要となる

例:
```
int factorial(int)
float pow(float, float)
```

9.2.4 スコープ(適用範囲)

スコープから見た引数の種類

外部変数(大域変数、global変数)

局所変数(local変数)

例題: 次のmain文の変数nはfactorial関数の呼び出しの前後で値が変わるか？

```
int main(int argc, char *argv[]) {
    int n,x;
    n = atoi(argv[1]);
    x = factorial(n);
   printf(”%d! = %d¥n” n,x);
}
```

```
int factorial(int x) {
    int i,n=1;
    for(i=1; i<=x; i++) n*=i;
    return n;
}
```

9.2.5 関数とマクロの違い

今までに

```
#define NUM 100
```

という「マクロ」を使ってきた。これはプログラムにおいてNUMという名前が出てきたら、すべて 100 で「置換」するということを意味するものであった

マクロにおいて、次のように数以外のものを書いても良い：

```
#define forever while (1)
```

これにより、プログラムにおいて次のように書くと、

```
forever { printf("I love you!¥n"); }
```

```
while (1) { printf("I love you!¥n"); }
```

として実行される

9.3 関数呼び出しの仕組み

9.3.1 関数とメモリ

C言語の関数は、CPUが理解できる命令(機械語)で書かれたプログラムとしてメモリに置かれ、それが実行される

関数名は、その実行開始アドレス(エントリーポイント）を表すシンボル

図9.6 参照

特に　main はプログラム全体のエントリーポイント

9.3.2 関数呼び出しの実際

関数を呼び出すとき

例: (組み込み) printf(“....”);

(自作関数) factorial(3);

このカッコは「関数を呼び出す演算子」

関数に渡される引数の値を計算し、スタックに積む

比較: count = 10; // 整数型変数 count

count(ar, 0.0, 1.0); // 自作関数countを3つの引数を与えて呼び出す

9.3.3 関数呼び出しとスタック

関数を呼び出すとき

(実)引数の値を計算し、その値を関数の(仮)引数の値としてセットする

計算された値はスタックに積まれる

値の渡し方: 基本は「値渡し」

これ以外に「アドレス渡し」「参照渡し」がある

配列や構造体の先頭アドレスを渡すやり方

9.3.3 関数呼び出しとスタック(続)

関数からの戻り

関数呼び出しの時に、「関数から戻ってきた時に実行されるアドレス」もスタックに積まれる

関数の処理中でもスタックが利用されるため、元の処理に戻るときは、スタックに保存されたレジスタの値を復元した後「戻り値を特定のレジスタに格納」

整数型の戻り値なら 汎用レジスタのどれか

浮動小数点数なら　浮動小数点数レジスタ